

Please type a plus sign (+) inside this box → ☒

PTO/SB/05 (12/97)

Approved for use through 09/30/00. OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

08/25/99

<b>UTILITY PATENT APPLICATION TRANSMITTAL</b> <small>Only for new nonprovisional applications under 37 CFR 1.53(b)</small>	Attorney Docket No.	21336-703	Total Pages	35 32
	First Named Inventor or Application Identifier			
	Peter H. van der Veen; Title: <i>Symmetric Multiprocessor System And Method</i>			
	Express Mail Label No.	EL324593814 US		

APPLICATION ELEMENTS <small>See MPEP chapter 600 concerning utility patent application contents.</small>	ADDRESS TO: Assistant Commissioner for Patents Box Patent Application Washington, DC 20231
1. <input checked="" type="checkbox"/> Fee Transmittal Form <i>(Submit an original, and a duplicate for fee processing)</i>	6. <input type="checkbox"/> Microfiche Computer Program <i>(Appendix)</i>
2. <input checked="" type="checkbox"/> Specification <span style="float: right;">[Total Pages <u>18</u>]</span> <i>(preferred arrangement set forth below)</i> <ul style="list-style-type: none"><li>- Descriptive title of the Invention</li><li>- Cross References to Related Applications</li><li>- Statement Regarding Fed sponsored R&amp;D</li><li>- Reference to Microfiche Appendix</li><li>- Background of the Invention</li><li>- Brief Summary of the Invention</li><li>- Brief Description of the Drawings</li><li>- Detailed Description</li><li>- Claim(s)</li><li>- Abstract of the Disclosure</li></ul>	7. Nucleotide and/or Amino Acid Sequence Submission <i>(if applicable, all necessary)</i> <ul style="list-style-type: none"><li>a. <input type="checkbox"/> Computer Readable Copy</li><li>b. <input type="checkbox"/> Paper Copy (identical to computer copy)</li><li>c. <input type="checkbox"/> Statement verifying identify of above copies</li></ul>
3. <input checked="" type="checkbox"/> Drawing(s) (37 CFR 1.152) <span style="float: right;">[Total Sheets <u>7</u>]</span>	8. <input checked="" type="checkbox"/> Assignment Papers (cover sheet & documents(s)) (3 pages)
4. <input checked="" type="checkbox"/> Oath or Declaration <span style="float: right;">[Total Pages <u>3</u>]</span> <ul style="list-style-type: none"><li>a. <input checked="" type="checkbox"/> Newly executed (original or copy)</li><li>b. <input type="checkbox"/> Copy from a prior application (37 CFR 1.63(d)) <i>(for continuation/divisional with Box 17 completed)</i><ul style="list-style-type: none"><li>[Note Box 5 below]</li><li>i. <input type="checkbox"/> <b>DELETION OF INVENTOR(S)</b> Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).</li></ul></li></ul>	9. <input type="checkbox"/> 37 CFR 3.73(b) Statement <input type="checkbox"/> Power of Attorney <i>(when there is an assignee)</i>
5. <input type="checkbox"/> Incorporation By Reference <i>(useable if Box 4b is checked)</i> The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.	10. <input type="checkbox"/> English Translation Document <i>(if applicable)</i>
	11. <input type="checkbox"/> Information Disclosure Statement (IDS)PTO-1449 <input type="checkbox"/> Copies of IDS Citations
	12. <input type="checkbox"/> Preliminary Amendment
	13. <input checked="" type="checkbox"/> Return Receipt Postcard (MPEP 503) <i>(Should be specifically itemized)</i>
	14. <input type="checkbox"/> Small Entity Statement(s) <input type="checkbox"/> Statement filed in prior application, Status still proper and desired
	15. <input type="checkbox"/> Certified Copy of Priority Document(s) <i>(if foreign priority is claimed)</i>
	16. <input type="checkbox"/> Other: .....

17. If a CONTINUING APPLICATION, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No. \_\_\_\_\_/\_\_\_\_\_

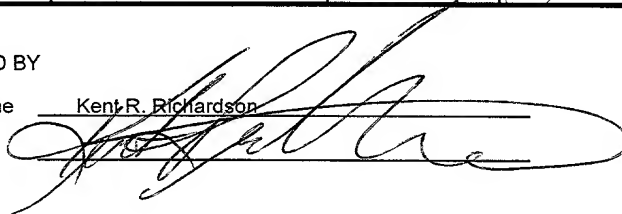
**16. CORRESPONDING ADDRESS**

<input type="checkbox"/> Customer Number of Bar Code Label	 <small>(Insert Customer No. or Attach bar code label here)</small>	or <input checked="" type="checkbox"/> Correspondence address below
NAME	Kent R. Richardson	
	Registration No.: 39,443 <span style="float: right;">PATENT TRADEMARK OFFICE</span>	
ADDRESS	WILSON SONSINI GOODRICH & ROSATI	
	650 Page Mill Road	
CITY	Palo Alto	STATE California
	ZIP CODE 94304-1050	
COUNTRY USA	TELEPHONE (650) 493-9300	FAX (650) 845-5000

SUBMITTED BY

Typed or Printed Name Kent R. Richardson

Signature



Reg. Number 39,443

Date

8/25/99

**Abstract**

Applicant: **QNX Software Systems Limited**

Our Reference: O8-880387CA

Date Printed: August 26, 1998

## ABSTRACT

The present invention relates generally to computer operating systems, and more specifically, to operating system calls in a symmetric multiprocessing (SMP) environment. Existing SMP strategies either use a single lock or multiple locks to limit access to critical areas of the operating system to one thread at a time. These strategies suffer from a number of performance problems including slow execution, large software and execution overheads and deadlocking problems. The invention applies a single lock strategy to a micro kernel operating system design which delegates functionality to external processes. The micro kernel has a single critical area, the micro kernel itself, which executes very quickly, while the external processes are protected by proper thread management. As a result, a single lock may be used, overcoming the performance problems of the existing strategies.

093315 03359  
665230 STE660

The present invention relates generally to computer operating systems, and more specifically, to operating system calls in a symmetric multiprocessing (SMP) environment.

## 5      **Background of the Invention**

Generally, computer systems are designed to accept and execute various application programs provided by a User, using an operating system to manage the computer resources required to execute the application programs.

665230" 516359  
10      Computer systems, and multiprocessor computer systems in particular, may manage execution of application programs by grouping program steps into "threads". Multiprocessor systems may assign different threads from the same application program to be executed on different processors to optimise use of the available resources. Groups of program steps are more easily managed as threads rather than single program steps which would create very large management overheads.

15      In addition to the optimal groupings of program steps, these threads contain parameter values that ensure the threads are executed within the proper time, are properly synchronized with other threads, or satisfy other constraints of the system. Real time operating systems, for example, will force threads to be executed before certain time deadlines so that real time interaction is maintained.

20      Symmetric multiprocessing (SMP) is the processing of application programs and operating systems using multiple processors that share a common operating system and memory. A single copy of the operating system is available to all the processors who share equal access to a common memory via a communication path of some form. SMP systems may also be described as "tightly coupled"

25      multiprocessing or "shared everything" systems.

30      The goal of an SMP system is to dynamically balance the workload between the available processors, optimising the use of the resources and providing the User with faster operation. Adding more processors to an SMP system results in faster operation, though completely linear improvement is not attainable because there are always critical sections of the Operation System that can only be executed one at a time. With completely linear improvement, two processors would run twice as fast as a single processor and three processors would run three times as fast as a single processor.

Because only a single copy of the operating system is available to all of the processors, two or more threads may attempt to access the same area of the operating system at the same time. Certain critical areas of the operating system will only be able to handle access by a single thread, otherwise causing unexpected or erroneous data to result. To prevent this, "locks" are commonly used to limit access to these critical areas to one processor or thread at a time. In order to access a critical area of the operating system, the thread must obtain the necessary lock or locks, and once it has completed execution, may make these locks available to other threads.

There are two common strategies to lock management: use of a single or global lock which locks the entire operating system, and use of multiple small locks which divide the operating system into isolated sections and lock each section separately.

In general, implementation of a single lock results in the timing shown in **Figure 1**. If all three **Processors 1, 2** and **3** require access to the operating system at the same time, only one will be given access and the other two must remain idle. Clearly, this strategy results in the overall system operating no faster than the speed of a single processor while operating system calls are being made. Because individual processors may operate uninhibited when they are not making operating systems calls, this strategy is still faster than a single processor.

More than one processor can access the operating system if different critical areas of the operating system are identified and a separate lock provided for each of these areas. Referring to **Figure 2**, an example of the timing of this second strategy is presented. In this example, all three **Processors 1, 2** and **3** require access to the operating system at the same time, but **Processors 1** and **2** require access to area A of the operating system, while **Processor 3** requires access to area B. Therefore, **Processor 3** is free to execute its call to the operating system independent of **Processors 1** and **2**, which must access the operating system at different times because there is only a single lock available for each operating system area. The Microsoft NT operating system is an example of an operating system applying such a multiple lock strategy.

This multiple lock strategy suffers from a number of performance problems. As the number of locks increases, the code complexity of the lock management software and number of locks to be acquired by a given thread increases. For

example, a single thread may require access to several areas of the operating system, requiring it to wait for all of the necessary locks to be made available. A thread may have obtained some, but not all of the locks it requires. While it is blocked, awaiting other locks to free up, it will be blocking any other threads waiting for the locks it has already obtained. This situation can result in deadlocking, where the computer system freezes because two threads are holding and waiting for each other's locks.

Even without deadlocking, this multiple lock strategy causes more time to be spent in searching and acquiring locks, and increases the difficulty of ensuring reliability.

There is also a commercial advantage to providing an operating system that is straightforward, as it is easier for driver programmers to write programs for the operating system. Use of multiple locks increases the complexity of the operating system, making the writing of driver programs more complex and time consuming, and the operation less predictable.

The most common architecture of SMP Operating Systems is described as a monolithic operating system. Monolithic operating systems incorporate most of the functionality of the operating system into a single program file, including input and output routines. Microsoft Windows CE is an example of such a monolithic operation system.

Because of the large size of monolithic operating systems, operating system calls typically take a long period of time to execute. Therefore a single lock strategy will generally result in unacceptably long time delays to the SMP system. Because of these long delays, a multiple lock strategy is preferred over the single lock for a monolithic operating system, but suffers from the same performance problems and deadlocking hazards outlined above.

SMP systems are commonly used in personal computer and networked computer systems, high-capacity telecom switches, image processing, and aircraft simulators. As well, SMP allows customers to extend the life and increase the cost-effectiveness of their multiprocessor systems by adding processor cards and their computing power to their multiprocessors rather than buying more systems.

There is therefore a need for a method and system of Symmetric Multiprocessing that addresses the problems outlined above. This design must be

provided with consideration for speed of execution, reliability, complexity and scalability.

### Summary of the Invention

5 It is therefore an object of the invention to provide an improved system and method of scheduling threads in a symmetric multiprocessing operating system.

One aspect of the invention is broadly defined as a method of symmetric multiprocessing in which one or more processors, a first memory medium storing a micro kernel operating system in a machine executable form and a second memory  
10 storing a thread scheduler in a machine executable form are interconnected via a communication network, the method comprising the steps within the thread scheduler of: responding to a thread requiring a call to the micro kernel operating system by requesting a global lock; and responding to the global lock being available by performing the steps of: acquiring the global lock from the thread scheduler;  
15 performing the call to the micro kernel operating system; and releasing the global lock.

### Brief Description of the Drawings

These and other features of the invention will become more apparent from  
20 the following description in which reference is made to the appended drawings in which:

- Figure 1** presents a timing diagram of a single lock strategy as known in the art;  
**Figure 2** presents a timing diagram of a multiple lock strategy as known in the art;  
**Figure 3** presents a symbolic layout of a symmetrical multiprocessor system in a  
25 manner of the invention;  
**Figure 4** presents a flow chart of a global lock management routine in a manner of the invention;  
**Figure 5** presents a symbolic layout of a symmetrical multiprocessor system identifying external operating system processes in a manner of the invention;  
30 **Figure 6** presents a timing diagram of the method of the invention; and  
**Figure 7** presents a flow chart of a global lock management routine in the preferred embodiment of the invention.

### Detailed Description of Preferred Embodiments of the Invention

The invention may be described with respect to the general symmetric multiprocessor (SMP) layout **10** shown in **Figure 3**. This figure presents an SMP layout consisting of a number of processors **12**, **14**, **16** and **18**, operable to execute application programs, a memory medium storing an operating system **20**, and a memory medium storing a thread scheduling program **22**. This computer system **10** may also include a variety of peripherals such as a printer **24** and a scanner **26**. These devices **12** through **26** may communicate with one another via a software bus **28**.

The processors **12**, **14**, **16** and **18**, may include personal computers, servers, micro controllers or other processing elements. Generally, each physical device on the system **10** is identified by a node address, so operation is essentially transparent to the physical arrangement. For example, while processor **12** may be operate as a server, the SMP system **10** does not treat it any differently than the other three processors **14**, **16** and **18**. The processors **12**, **14**, **16** and **18**, are therefore described as "peers", each having equal access to the resources controlled by the operating system **20**.

If an individual computer has a number of processors within it, each having access to the software bus **28**, then each of these processors will also be considered to be a peer to any other processor accessible from the software bus **28**. Processors which do not have direct access to the software bus **28** will require administration by another operating system.

The software bus **28** may consist of any manner of communication network which allows software data to be communicated between the processors **12**, **14**, **16** and **18** and other components of the system **10**. The processors **12**, **14**, **16** and **18** may all reside on a single printed circuit board, in which case the software bus **28** may comprise copper tracks and the necessary input and output drivers. If the processors **12**, **14**, **16** and **18** comprise individual personal computers, then the software bus **28** may comprise electrical cable connections and communication hardware as known in the art.

The operating system **20** is generally stored in an executable form on a computer readable medium such as a random access memory (RAM), read only memory (ROM), optical disk (CD-ROM) or magnetic medium (hard drive or portable diskette). Of course, the operating system **20** could also be implemented by



hardware means, or other means known in the art. The operating system 20 is accessible to each of the processors 12, 14, 16 and 18, and generally provides such capabilities as interprocess communications, message passing, data input and output, and timing services.

5           The thread manager 22 is also generally stored in an executable form on a similar computer readable memory medium accessible to each device via the software bus 28. The thread manager is not usually stored with the operating system 20 or within a processor 12, 14, 16 or 18, so that it is accessible at any time.

10           A thread scheduling program 22 which addresses the objects outlined above may be described by the flow chart of **Figure 4**. This figure presents a method of symmetric multiprocessing in which one or more processors 12, 14, 16 and 18, a first memory medium storing a micro kernel operating system 20 in a machine executable form and a second memory storing a thread scheduler 22 in a machine executable form are interconnected via a communication network 28, possibly but  
15           not necessarily in the arrangement presented in **Figure 3**. The method starting at step 30 comprises the steps within the thread scheduler 22 of: responding to a thread requiring access to the micro kernel operating system at step 32, by requesting a global lock at step 34. If the thread does not require access to the operating system, regular thread management is invoked at step 36. For the thread  
20           which does require access to the micro kernel operating system, the method then responds to the global lock being available by performing the steps of: acquiring the global lock at step 38; executing the thread on the micro kernel operating system 20 at step 40; and then releasing the global lock at step 42. Once the global lock has been released at step 42, control returns to step 32 to handle another thread. Until  
25           the global lock is made available to the thread, the request for the global lock remains in a tight loop at step 34.

          In general, the invention requires that the SMP system take on a physical arrangement similar to that described above with respect to **Figure 3**. Clearly modifications can be made to this physical arrangement which still obtain the  
30           benefits of the invention. Such modifications would be clear to one skilled in the art. As well, the invention may provide some benefit to completely different processing systems such as Asymmetric Multiprocessing systems.

          The invention also requires, in a broad sense, that the operating system 20 be of a micro kernel architecture. A micro kernel operating system is one in the

operating system itself provides minimal services which delegate the usual operating system functions to external processes. These services are generally described as Inter-Process Control or IPC services.

An example of external processes which may be accessed by such a micro kernel is presented in **Figure 5**. In this example, the micro kernel operating system **20** is connected to the software bus as in **Figure 3**. However, additional external processes are also accessible via the software bus **28**, such as the DOS file manager **44**, CD-ROM file manager **46**, Graphic User Interface Manager **48** and TCP/IP Manager **50**. Operation and implementation of such processes are well known in the art, as are other processes which may be required. Optimal operation of the invention is obtained by implementing such processes external to the micro kernel operating system.

This modularity allows the relative size of the operating system to be greatly reduced, increasing speed of execution. Although a micro kernel is generally smaller than a monolithic operating system, it is the modularity that results in the relative improvement in speed, and allows the method of the invention to be used.

This modularity also allows the method of the invention to execute SMP much faster than the methods known in the art. **Figure 6** presents a timing diagram which compares the method of the invention to the methods described in the Background of the Invention per **Figures 1 and 2**. In **Figure 6** all three **Processors 1, 2 and 3** require access to the operating system at the same time. Because the operating system is a micro kernel system, the only critical area of the operating system is in the kernel, and not in the external routine calls. The external processes are executed as priority based threads, so if the threads are scheduled properly using known techniques, there is no danger of unexpected results or errors. Therefore, a thread need only obtain the global lock to perform the kernel call, and may release it before completing the operating system call in an external process.

In **Figure 6** each **Processor 1, 2 and 3** is shown to execute a kernel call designated by a "K", and an "External call". None of the kernel calls may overlap, but the external routines may. Since the kernel calls are so short, taking only microseconds to execute, the SMP system executes faster than the systems known in the art.

The relative time lengths of the calls in **Figure 6** are symbolic only. In actual practice, the external processes may not execute immediately after the kernel call if

they are not required to. As well, the external processes may take hundreds or thousands of times longer to execute than the kernel call. However, the micro kernel may be designed to have a specific and predictable execution time, allowing the method of the invention to guarantee real time performance.

5           The degree of modularity of the operating system will depend on the processes required, and on other performance considerations. In certain applications for example, only a small number of external processes may be required. If only one or two processes are required, it may be advantageous to keep these processes with the micro kernel. Though not a micro kernel in the true sense,  
10       such an arrangement may execute a compromised version of the invention.

          Clearly, the faster the micro kernel operating system executes operating system calls with respect to the exercise of collecting multiple locks, the more efficient the method of the invention will be. With consideration for the other advantages of a single lock strategy, the single lock is preferred over the multilock  
15       strategy when the time spent in the operating system call is less than the time spent searching and acquiring a large number of small locks.

          Broadly speaking, the invention may be implemented by modifying known lock and thread management routines to schedule thread execution in the manner of the invention.

20           Determination as to whether a thread requires access to the operating system 20 at step 32 may be done in a number of manners as known in the art. For example, the thread scheduler may identify each process that a thread requires and set a boolean flag to indicate the micro kernel operating system call.

          A simple compare and swap sequence may be used to obtain and release  
25       locks. Since only a single lock is being used, it is easy to set a boolean flag to indicate whether the global lock is available at step 34. Generally, locks are handed out on a FIFO (first in, first out) basis, that is, that the first thread requesting a global lock receives it. Conflicts in lock requests, that is, two or more threads requesting the global lock at the same time, must also be handled. Various techniques for  
30       handling such conflicts are known in the art, such as distinguishing the priority of Operating System calls by the nature of the call, preferring those calls which impact the real time operability of the system.

          As noted above, acquiring the global lock at step 38 may be handled simply by setting a boolean flag in the thread manager, or by passing a token to the thread

with the kernel call. Both techniques and others, are well known in the art. The step of releasing the global lock at step 42 may be implemented in a complementary manner to that used to implement step 38.

5 Execution of the critical thread at step 40 may be performed in the manner known in the art, and particular to the operating system upon which the invention is applied. No additional functionality is required to implement this step in the manner of the invention.

10 It should be noted however, that some operation systems use a "spin lock" to protect the operating system while it is executing. This is not to be confused with the global lock of the invention which is used manage threads. The implementation of spin locks in protecting an executing operating system is well known in the art.

Step 36 of course, may also be implemented in the manner particular to the operating system upon which the invention is being implemented.

15 Therefore, the invention adds a small amount of overhead to access the kernel but once accessed, the code paths are all the same.

20 This method provides for execution of a Symmetric Multiprocessor system with a single lock. Having one lock for the whole micro kernel requires minimal time to administer, and only adds a small amount of code. Micro kernels minimize system calls by delegating to external processes which may be executed in other processors. As a result, the call to the operating system kernel takes very little time.

Because only a single lock is required, the system is able to identify the status of and acquire the lock very quickly, and there is no danger of deadlocking as in the case of multiple locks.

25 In contrast to monolithic operating systems, micro kernel architectures distribute the functionality of the operating system into separate files as much as possible, leaving only a kernel with message passing capabilities to administer the operating system. All file handling, input and output handling, and other operations are provided in external processes.

30 Because traditional monolithic kernels contain the bulk of all operating system services, they require numerous performance-robbing spin-locks in the main code paths to support SMP. In contrast, the invention uses a lean micro kernel architecture requiring only a single lock, resulting in faster performance.

Simpler implementation of SMP leads to fewer bugs and better performance. Complete linear speed up with added processors, is the ultimate goal of an SMP

system, but can not be attained because there are always critical sections of the operating system that can only be executed one at a time. However, the method of the invention provides a method which will tend towards linear speed ups with reductions in the size of the micro kernel, reducing the length of the critical code to execute.

In general, a single lock SMP can only be efficiently applied to a micro kernel operating system and not to a monolithic operating system. As indicated in **Figure 1** and described above, a single lock SMP applied to a monolithic operating system results in poor performance due to the long monolithic kernel calls. The lock must wait until the kernel call has finished executing before it could take control. While waiting for the long kernel call to finish, the additional processors are idle.

However, a monolithic operating system may obtain some of the benefit of the invention by minimizing the code in the kernel and moving functionality to external locations. Message passing functions would be required for the operating system to access external processes, but such techniques are known in the art. This reduction in the size of the operating system kernel would reduce the time required to execute operating system calls, allowing the invention to be applied.

In the preferred embodiment, it is intended to implement the invention as a generic and commercially available product, as opposed to customized. The SMP functionality may be implemented as an add on to the basic operating system software, giving the User the option of either uniprocessor or SMP implementation. Basically only one file handling the global lock would have to be enabled to change from uniprocessor operation to SMP. The balance of the operating system and the user processes are unchanged, and have no knowledge of which mode the system is operating.

The most common implementation would be for an Intel X86 platform, though the invention may be applied to any platform by changing the process to suit the necessary platform API. Implementation on MIPS or PPC for example, can easily be made by modifying the SMP hardware and operating system to interact with the hardware of the new platform.

In the preferred embodiment, the invention will be applied to a real time operating system (RTOS). In a real time operating system it is required that certain functions be executed within certain periods of time. Therefore, to implement the invention, it may be necessary to prioritize operating system calls in order of how

their respective call latencies impact the real time operation. This would for example, allow the kernel calls which delay the User the most, to be executed first.

As will be described with respect to **Figure 7**, such an implementation requires that execution of non-critical threads may be pre-empted so that critical threads may be executed on the operating system. "Pre-emption" is the act of one thread substituting itself for another executing, dispatchable thread on a processor. For example, if a machine interrupt is received while a thread with an operating system call is holding the global lock and executing, the processor must go into an idle loop while the machine interrupt is handled, then returns and continues doing the spin. In a real time system, it is generally presumed that the machine interrupt is of the highest priority, so the delay of the kernel call being pre-empted is of secondary concern to the execution of the machine interrupt.

Implementation of the invention with pre-emption provides a further performance improvement over the prior art, by allowing real time operations to pre-empt non-real time operations.

**Figure 7** presents a flow chart of the method of the preferred embodiment. Again, a physical arrangement similar to **Figures 3 or 5** is suitable for such a method, though it is not necessary to have the same components or physical arrangement.

The method starting at step **52** builds on the method described with respect to **Figure 4**. In this case, the thread scheduler responds to a thread requiring access to the pre-emptable micro kernel operating system at step **54**, by requesting the global lock at step **56**, while non-critical threads are managed in the manner known in the art at step **58**. The thread scheduler responds to the global lock being available at step **56** by determining whether non-critical threads are currently running on the operating system **20** at step **60**. If no such threads are currently executing, then the global lock may be acquired at step **62**, otherwise the non-critical threads must be pre-empted at step **64**.

As noted above, in the preferred embodiment, real time operation is provided by the pre-emption of lower-priority threads currently being executed. Such threads are therefore pre-empted at step **64** before the global lock is acquired. Existing thread-based operating systems have the functionality to handle machine interrupts which suspend execution of a thread to handle the interrupt. In a manner of the invention, a similar routine may be used to pre-empt execution of a thread to allow

execution of the thread requiring access to the operating system. The routine which performs the pre-empting may also set a boolean flag to indicate that threads were pre-empted.

5       Once the global lock has been acquired at step 62, the thread call to the operating system 20 may be executed at step 66. The requires of this call will depend on the platform upon which the operating system is running, and the nature of the operating system itself. Briefly, the kernel call in the preferred embodiment will comprise execution of the following steps:

1.       Entry into the Kernel

10       This step includes execution of code required to pass the thread call into the kernel for execution. As noted above, some operating systems may require a special spin lock to be acquired as part of this step, to protect the operating system. This spin lock is different from the global lock which is being used as a management tool. Before leaving this step, the spin lock may be released.  
15       During this step, no pre-emption is allowed and interrupts are not handled.

2.       Prologue

20       During this step, the necessary initializations are being executed to set up the kernel operation at step 4. No kernel data structures may be modified during this step. However, the kernel may access or verify that it can access user data during this step. This step is fully pre-emptable, and interrupts may be handled.

3.       Kernel Operation

25       This step includes execution of the thread call to the operating system kernel. As described above, this call may comprise one of a number of file handling, data input or output, or other operating system functions. Being a micro kernel design, the bulk of the code required to handle these functions is resident in external processes, so the kernel execution generally just sets up the external process call for execution. This is the only step of the kernel call during which kernel data structures may be modified. During this period the  
30       operating system is not pre-emptable, though interrupts may be handled.

4.       Epilogue

      This optional step is used to complete the execution of the kernel call. Similar to the prologue of step 2 above, user data may be accessed during this step, or verification can be made that user data may be accessed.

During this period the operating system is fully pre-emptable and interrupts may be handled.

5. Exit

This stage performs the return of the thread, or possibly an error message, from the operating system kernel to the processor executing the thread.

Similar to the Entry step 1 above, the spin lock may be acquired at the beginning of this step and released at the end, to protect the operating system. No pre-emption is allowed during this stage, and no interrupts may be handled.

To summarize the accessibility of the kernel during step 66:

Step	Pre-emption	Interrupts
1. Entry	Not allowed	Not handled
2. Prologue	Allowed	Handled
3. Kernel Operation	Not allowed	Handled
4. Epilogue (optional)	Allowed	Handled
5. Exit	Not allowed	Not handled

The global lock may then be released at step 68. As noted above, this may be done in a complementary manner to the method used to acquire the lock at step 62.

Determination is then made at step 70 as to whether any threads were pre-empted at step 64. This determination may be made on the status of a boolean flag indicating that pre-emptions were made, or by the existence of thread identities and parameters in an array used to store the pre-empted threads. If no threads were pre-empted, then the routine returns to step 54, otherwise pre-empted threads must be reinstated at step 70. Reinstatement of these threads may be performed in the manner that complements how the threads were pre-empted at step 56.

As described with respect to the general case herein above, known global lock and thread management routines may be modified to effect the invention.

The invention may also be applied with further optional features as known in the art, such as:

1. Returning threads to the processor they ran on previously to optimize memory cache performance.



2. Use of processor "affinity masking" to select which processor each thread may run on, further optimizing performance.
3. Use in embedded systems. Because of the small amount of memory required, scalability to add functionality, and performance, the invention may be applied to embedded applications.
4. Routing machine interrupts to the processor handling the lowest priority threads.
5. Adding a spin lock acquire and release around the Prologue step 2 and/or Epilogue step 4 allowing multiple calls to be in the Prologue at the same time. However, if a call was in the Kernel Operation step 3 at the time, a new call would not be able to access the Prologue step 2 or Epilogue step 4.

While particular embodiments of the present invention have been shown and described, it is clear that changes and modifications may be made to such embodiments without departing from the true scope and spirit of the invention. For example, modifications to larger or monolithic operating systems could be made to apply the teachings of the invention and realize performance improvements. As well, hybrids of the thread management system of the invention with existing management strategies may be appropriate to particular thread sizes or applications.

The operating system of the invention could be embedded into a micro controller, digital signal processor or intelligent instrumentation, operating as a piece of electronic hardware or as part of the electronic system. The invention could also be implemented in the form of machine executable software; the method of the invention being transmitted as electronic signals, or stored in machine readable or executable form in random access memory (RAM), read only memory (ROM), optical disk (CD-ROM) or magnetic storage media (hard drive or portable diskette).

An operating system in a manner of the invention could be applied to a broad range of applications, including stand-alone uniprocessor systems, multiprocessor or network-connected systems, servers, palm top or laptop computers, cellular telephones, automobile controllers and smart process control instruments. Again, such implementations would be clear to one skilled in the art, and does not take away from the invention.

Since the invention offers the best possible utilization of available Processor cycles, it's ideal for very high-end real-time applications such as high-capacity telecom switches, image processing, and aircraft simulators.

WHAT IS CLAIMED IS:

1. A method of symmetric multiprocessing in which one or more processors, a first memory medium storing a micro kernel operating system in a machine executable form and a second memory storing a thread scheduler in a machine executable form are interconnected via a communication network, said method comprising the steps within said thread scheduler of:  
responding to a thread requiring a call to said micro kernel operating system  
by requesting a global lock;  
responding to said global lock being available by performing the steps of:  
acquiring said global lock from said thread scheduler;  
performing said call to said micro kernel operating system; and  
releasing said global lock.
2. A method of symmetric multiprocessing as claimed in claim 1, wherein said micro kernel operating system comprises a pre-emptable micro kernel operating system, said method further comprising the steps with said thread scheduler of:  
pre-empting any non-critical threads currently executing on said pre-emptable micro kernel operating system prior to said step of executing said thread on said pre-emptable micro kernel operating system; and  
reinstating said pre-empted threads following said step of executing said thread on said pre-emptable micro kernel operating system.
3. A method of symmetric multiprocessing as claimed in claim 2 wherein said step of performing said call to said micro kernel operating system comprises the steps of:  
entering said pre-emptable micro kernel operating system;  
executing operating system functions as required by said thread;  
locking said pre-emptable micro kernel operating system; and  
exiting said pre-emptable micro kernel operating system.

4. A method of symmetric multiprocessing in which one or more processors, a first memory medium storing a pre-emptable micro kernel operating system in a machine executable form and a second memory storing a thread scheduler in a machine executable form are interconnected via a communication network, said method comprising the steps within said thread scheduler of:
  - responding to a thread requiring a call to said micro kernel operating system by requesting a global lock;
  - responding to said global lock being available by performing the steps of:
    - pre-empting any non-critical threads currently executing on said pre-emptable micro kernel operating system;
    - acquiring said global lock from said thread scheduler;
    - entering said pre-emptable micro kernel operating system;
    - executing operating system functions as required by said thread;
    - locking said pre-emptable micro kernel operating system;
    - exiting said pre-emptable micro kernel operating system;
    - releasing said global lock; and
    - reinstating said pre-empted threads.
5. A computer system comprising:
  - one or more processors;
  - a first memory medium storing a pre-emptable operating system in a machine executable form;
  - a second memory storing a lock manager in an machine executable form;
  - a communication network interconnecting said one or more processors, said first memory and said second memory; and
  - said lock manager being operable to:
    - responding to a thread requiring access to a critical area of said pre-emptable operating system by requesting a global lock;
    - responding to said global lock being available by performing the steps of:
      - pre-empting any non-critical threads currently executing on said operating system;
      - executing said critical thread on said operating system; and
      - reinstating said pre-empted threads.

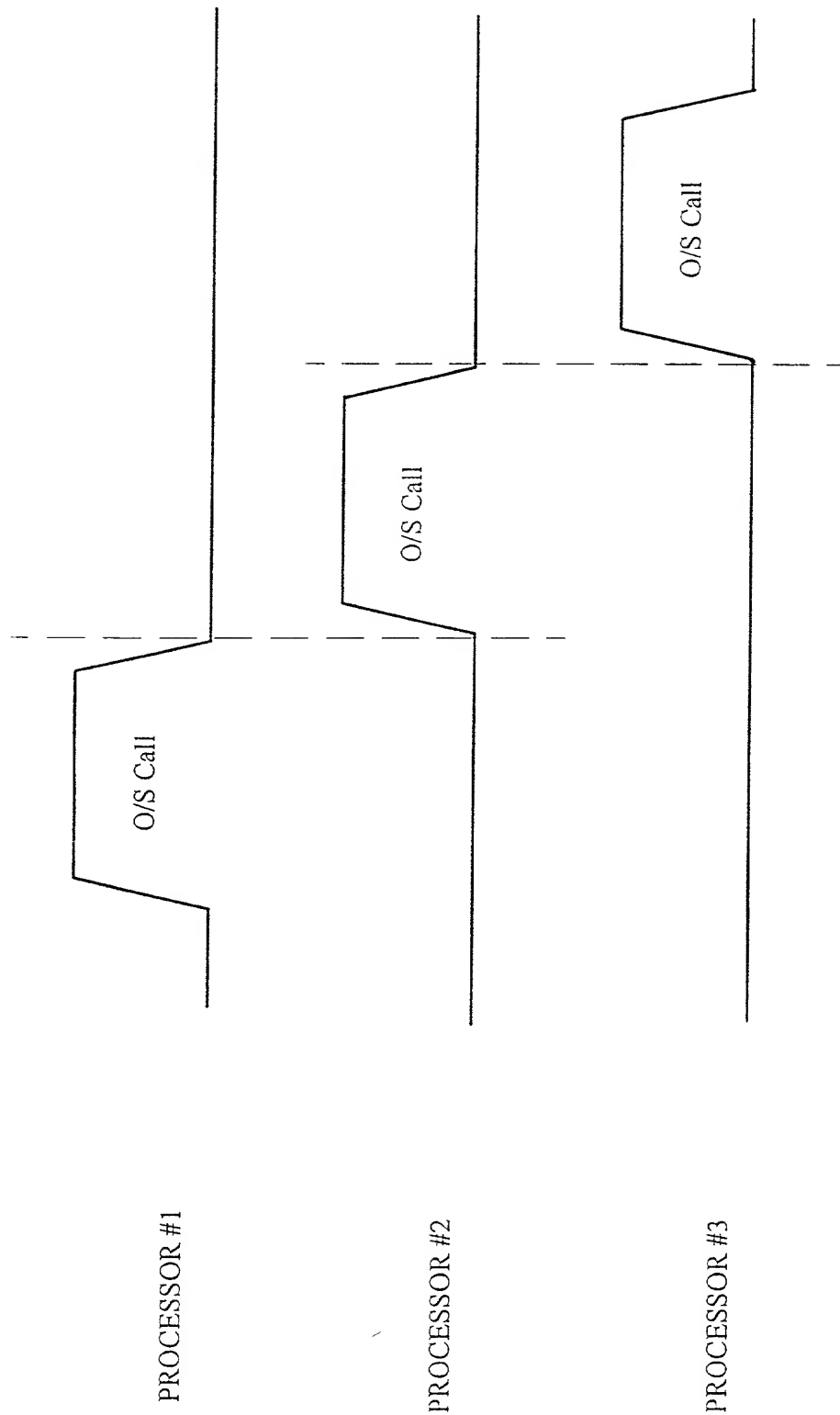


FIGURE 1 - PRIOR ART

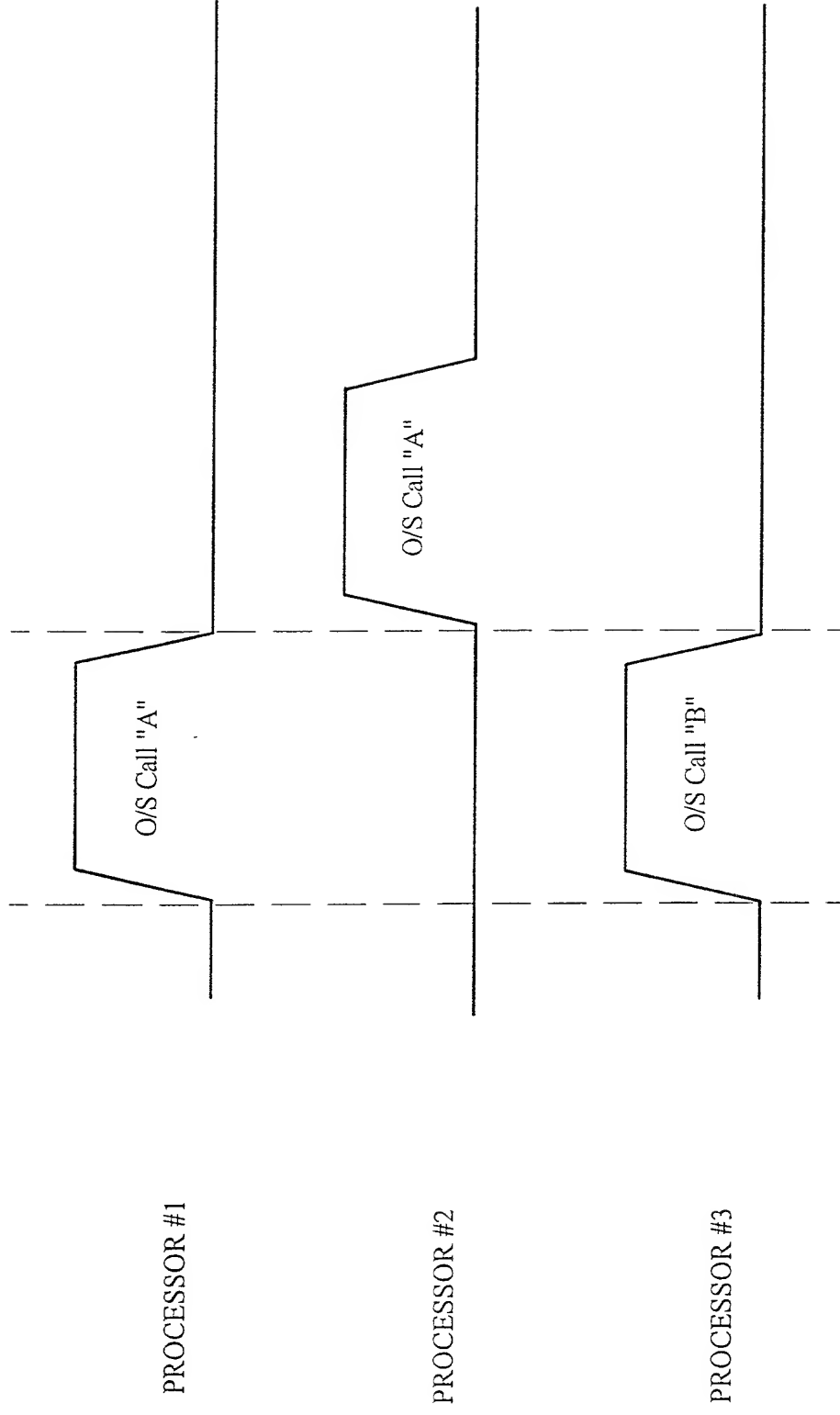


FIGURE 2 - PRIOR ART

FIGURE 3

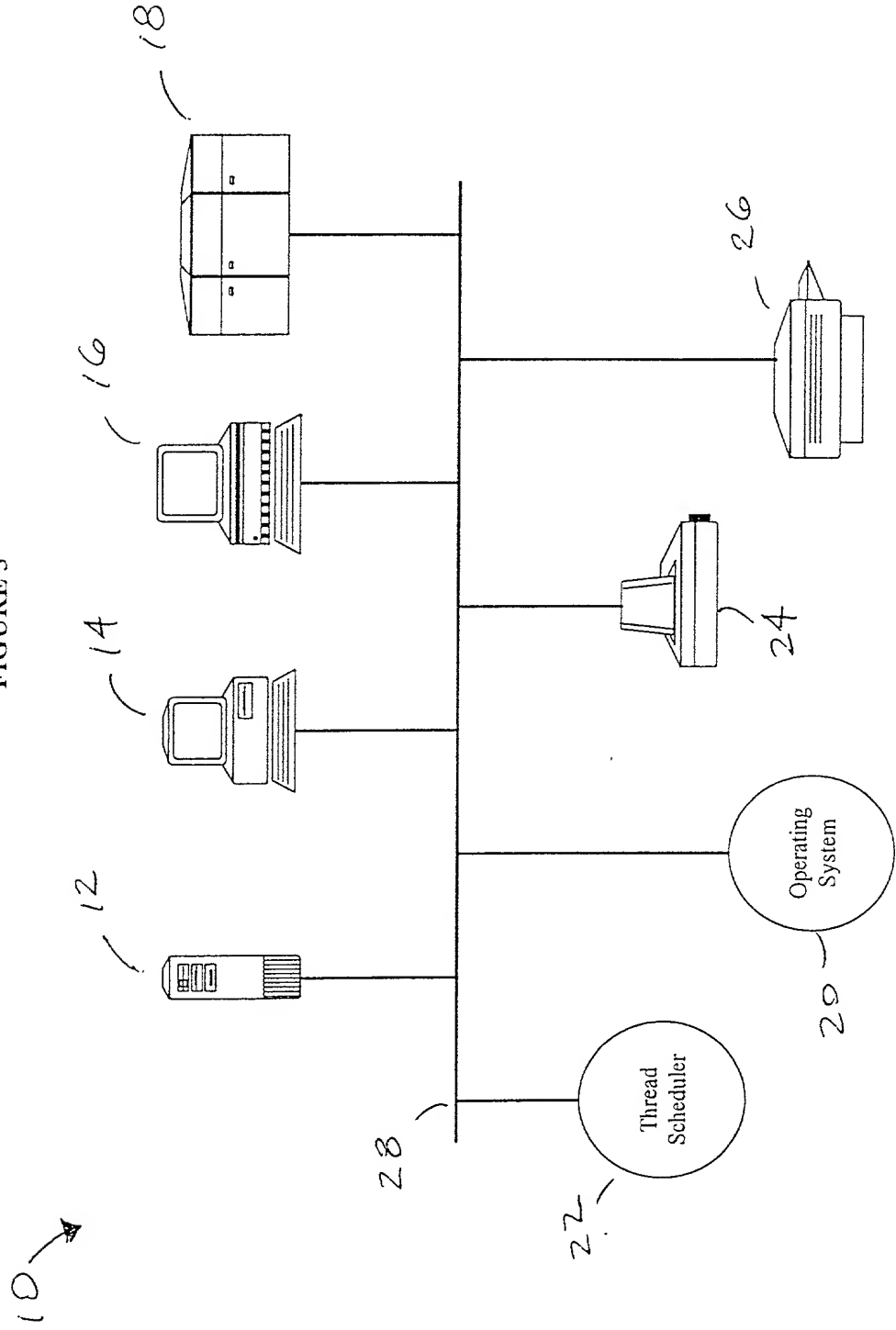


FIGURE 4

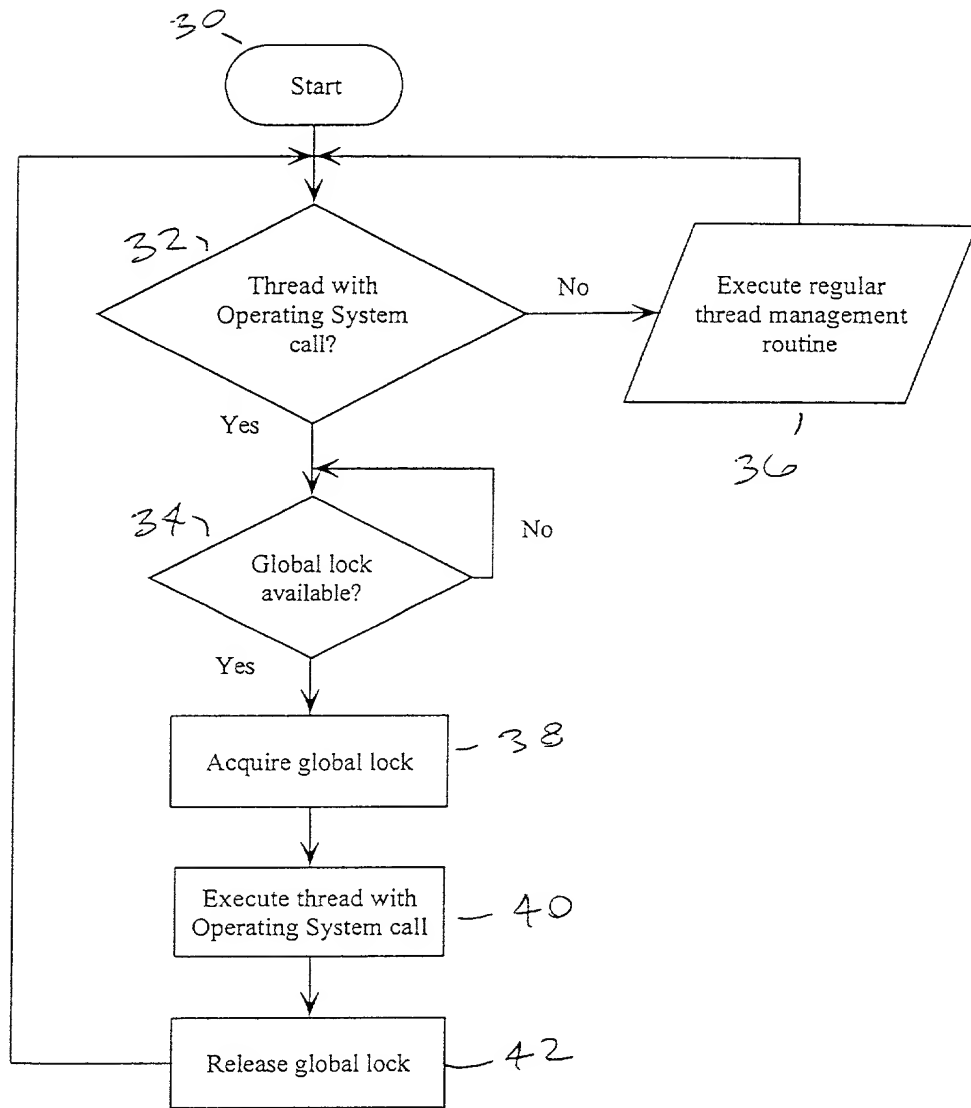
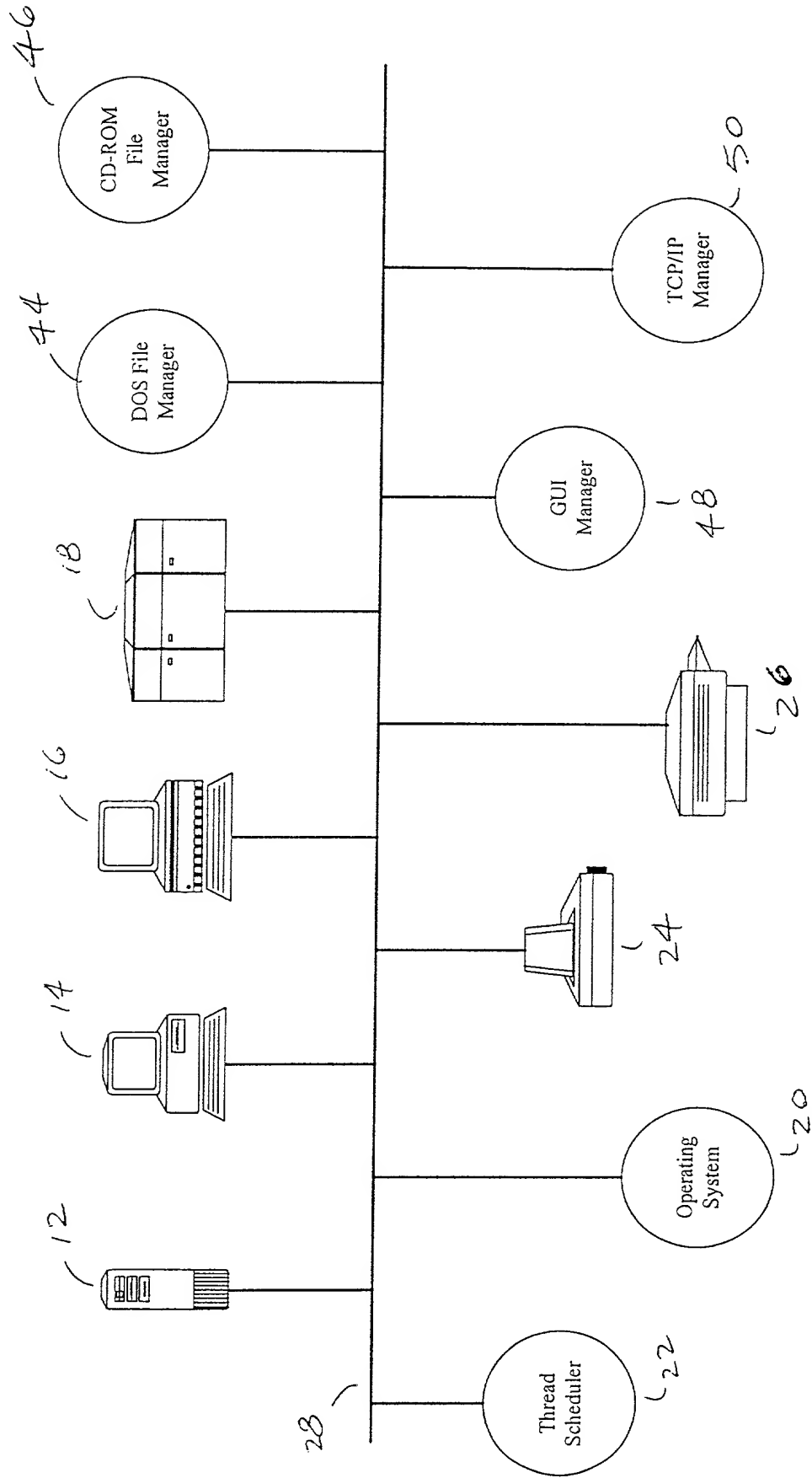


FIGURE 5





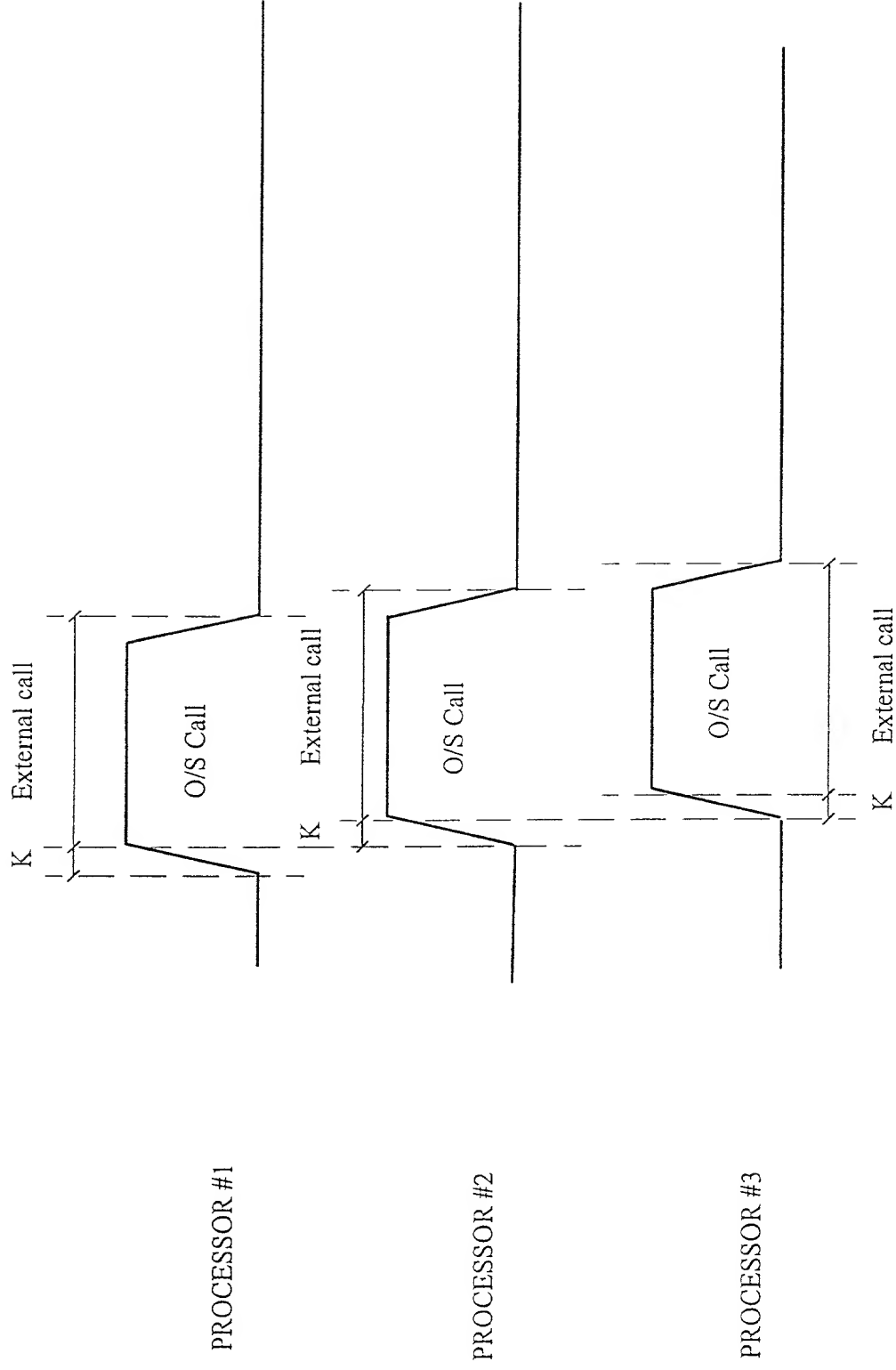


FIGURE 6

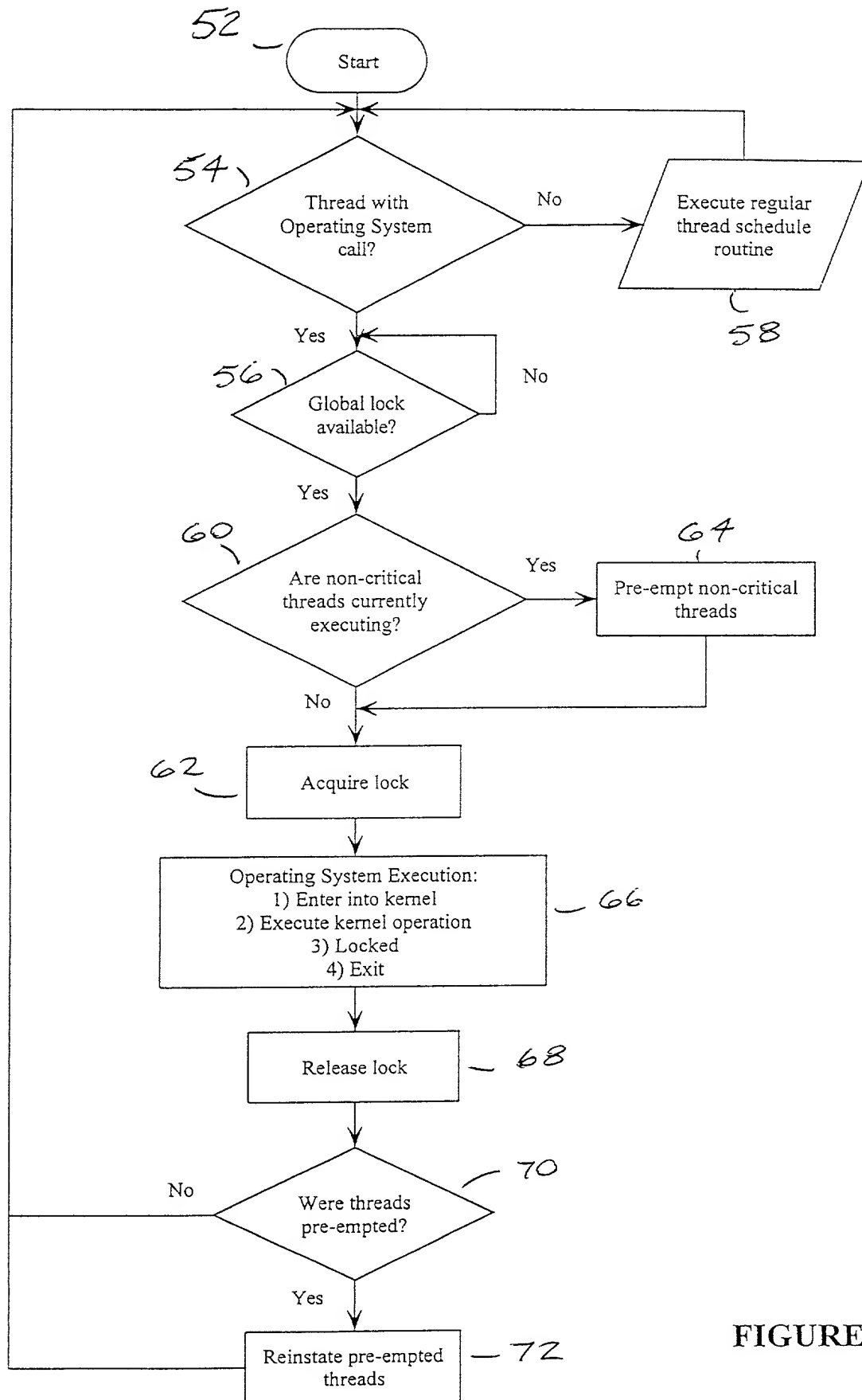


FIGURE 7

**COMBINED DECLARATION AND POWER OF ATTORNEY***ORIGINAL, DESIGN, NATIONAL STAGE OF PCT, SUPPLEMENTAL*

As a below named inventor, I hereby declare that:

**TYPE OF DECLARATION**

This declaration is of the following type: Original

**INVENTORSHIP IDENTIFICATION**

My residence, post office address and citizenship are as stated below next to my name, I believe I am the original, first and sole inventor (*if only one name is listed below*) or an original, first and joint inventor (*if plural names are listed below*) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

**Symmetric Multi-Processor System And Method****SPECIFICATION IDENTIFICATION**

the specification of which:

■ is attached hereto.

**ACKNOWLEDGEMENT OF REVIEW OF PAPERS AND DUTY OF CANDOR**

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations. § 1.56(a).

**PRIORITY CLAIM**

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed.

EARLIEST FOREIGN APPLICATION(S), IF ANY, FILED WITHIN 12 MONTHS  
(6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

Country	Application No.	Date of Filing dd/mm/yyyy	Priority Claimed Under 37 USC 119
Canada	2,245,976	26/08/1998	

ALL FOREIGN APPLICATION(S), IF ANY, FILED MORE THAN 12 MONTHS  
(6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

Country	Application No.	Date of Filing
---------	-----------------	----------------

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Application Serial No.	Filing Date	Status (patented, pending, abandoned)
------------------------	-------------	---------------------------------------

#### POWER OF ATTORNEY

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (*List name(s) and registration number(s)*)

Paul Davis - No. 29,294	Barbara Courtney - No. 42,442
John J. Bruckner - No. 35,816	Travis L. Dodd - No. 42,491
David J. Weitz - No. 38,362	Richard L. Gregory, Jr. - 42,607
Kent R. Richardson - No. 39,443	Van Mahamedi - No. 42,828
David J. Abraham - No. 39,554	Shantanu Basu - No. 43,318
U.P. Peter Eng - No. 39,666	Kenta Suzue - No. P45,145
George A. Willman - No. 41,378	Shaal Mehra - No. P44,934
Jinntung Su - No. 42,174	Joel Harris - No. P44,743
X. Shirley Chen - No. P44,608	

#### DECLARATION

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such wilful false statements may jeopardize the validity of the application or any patent issued thereon.

#### SIGNATURE (s)

Full name of first inventor: **van der VEEN, Peter H.**

Country of Citizenship: **Canada**

Residence Address: **105 Balbair Road, Kanata, Ontario, K2L 1G5, Canada**

Post Office Address: **Same as Residence Address**

Date:

24<sup>th</sup> August 1999

Signature:

Peter van der Veen